情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　56–66 (Sep. 2010)

動的推定によるプリフェッチ量最適化

入江英嗣[†1,*1]　本城剛毅[†1]　平木敬[†1]

プロセッサアーキテクチャにおける重要な課題として，増加を続けるメモリレイテンシの克服があげられる．データを投機的に先読みするプリフェッチ技術はレイテンシ隠蔽に効果的であり，汎用プロセッサへの採用が進んでいる．しかしプリフェッチには，プログラムやキャッシュ容量次第で最適プリフェッチ量が変化し，性能を大きく増減させるという問題がある．最適なプリフェッチ量を実現するために，予測精度などに着目してフィードバックをかける手法が提案されているが，プログラムとキャッシュ容量のバランスは多様であり，まだ十分な制御とはなっていない．本論文では，理想的なプリフェッチ量に関する考察をふまえ，**キャッシュの内部データ対流**と**データ取り込み速度**のバランスに着目したプリフェッチスロットリング，CCCPO（Cache-Convection-Control based Prefetch Optimization）を提案する．一般的なスロットリング手法と異なり，提案手法はプリフェッチ履歴を用いず，数本のカウンタで実装可能である．シーケンシャルプリフェッチに適用した評価では，多様なプログラムに対して安定したプリフェッチを実現した．予測精度に着目した従来スロットリング手法に対し，提案手法は最大で 13%，調和平均でも 1.3%の性能向上を示した．

Prefetch Throttling Technique Based on Dynamic Assumption

Hidetsugu Irie,[†1,*1] Goki Honjo[†1] and Kei Hiraki[†1]

One of the significant issues of processor architecture is to overcome increasing memory latency. Prefetches are commonly used in general propose processors because of their effectiveness for latency hiding. However, prefetches have a drawback that they can both increase and decrease performance unless their aggressiveness is set properly. Although several techniques which throttle prefetch-aggressiveness with some metrics like accuracy are proposed, their controls are not sufficient due to the variation of the balance between program working sets and cache capacities. In this paper, we discuss the ideal prefetch throttling, and based on this, propose CCCPO (Cache-Convection-Control based Prefetch Optimization) throttling, which controls the balance between "line convection" and input speed of the cache. Our technique is able to be implemented with few counters. Introduced into sequential prefetcher, our technique achieved stable prefetch for various programs. It showed max 13% performance improvement against the existing prefeth throttling which focused on prefech accuracy, and 1.3% in geometric mean.

1. はじめに

メモリレイテンシを隠蔽してコアの待機時間を減らすことは，マイクロアーキテクチャの主要課題である[1)]．特に，近年ではオフチップメモリレイテンシは数百サイクルに達し，多くても 200 命令程度の範囲でしか命令の実行順を入れ替えることができない，アウト・オブ・オーダ実行では軽減は難しい．プリフェッチ[2),3)]は投機的な手法であるが，簡単な予測アルゴリズムによってアクセスタイミングを先行させ，この数百サイクルのレイテンシを隠蔽することができる．メモリバンド幅への影響は比較的軽量で，またアプリケーションへの適用範囲が広い．メモリレイテンシの問題に対して，データ並列性やスレッド並列性を利用し，メモリアクセスをオーバラップさせるアーキテクチャ手法[4)]も有効であるが，計算時間と転送時間が釣り合わないと隠蔽効率が低下するため，現在ではまだ適用できるアプリケーションが限られている．

このような理由から，汎用プロセッサにおいてハードウェアプリフェッチ[2),3)]の重要性は増している．様々なプリフェッチ手法が商用プロセッサに搭載されるようになり[5)]，また，近年ではさらに進んだプリフェッチ手法の提案も次々に行われている[6)–9)]．

しかし，プリフェッチには，アプリケーションやキャッシュ容量によって，適切なプリフェッチ量が異なるという問題が存在する．積極的なプリフェッチは，一部のアプリケーションで有効に働く一方，別のアプリケーションではキャッシュから有効ラインを溢れさせ性能を激減させてしまう（キャッシュポリューション）．また，予測精度が低いと，無駄なリクエストが本来のデマンドアクセスの数倍以上に及ぶこともあり，メモリトラフィック量を増加させ，バス競合や DRAM バンク競合の原因となってしまう．特に，あらかじめ決められたアルゴリズムに従って一律に適用されるハードウェアプリフェッチでは，副作用の影響は大きい．

†1 東京大学大学院情報理工学系研究科
Graduate School of Information and Technology, the University of Tokyo
*1 現在，電気通信大学大学院情報システム学研究科
Presently with Graduate School of Information Systems, the University of Electro-Communications

このため，新しく提案されるプリフェッチャではプリフェッチ量の調整機能を持つことが一般的となってきている．これらの多くでは，キャッシュは貴重な資源という考え方から，精度を基準とした制御が提案されている[10)–12)]．プリフェッチしたラインに実際にアクセスがあったかを観測し，予測精度を求め，あらかじめ設定しておいた閾値を下回るようであれば，プリフェッチを抑制する．

しかし，近年ではキャッシュの大容量化から，複数候補に網を張るようなプリフェッチが成果をあげており，高精度は必ずしも高性能とは対応しない[9)]．その一方で，キャッシュ容量がワーキングセットサイズに近い場合，精度の高いプリフェッチでもポリューションを引き起こしてしまう可能性がある．このためプリフェッチの成績に基づいた制御では，制御の有効性は発見的に設定された閾値に強く依存してしまう．

そこで，本論文では，このような閾値に依存せずに，多様なプログラムについて安定したプリフェッチ量を実現する手法の提案を行う．最適なプリフェッチスロットリングについて考察を行い，従来手法では無視されていた，キャッシュの内部データ対流とデータ取り込み速度のバランスに注目したスロットリング手法，CCCPO（Cache-Convection-Control based Prefetch Optimization）を提案する．

以下，本論文は次のように構成される．2 章ではプリフェッチ量の調整技術について，従来手法を紹介しながら議論する．3 章ではキャッシュ内のデータ対流について考察し，この振舞いを観測することによって最適制御が可能であることを示す．4 章では提案機構の詳細について述べる．5 章で評価環境を示し，6 章で提案手法の評価と，従来手法との比較を行う．7 章で関連研究について紹介し，8 章でまとめを述べる．

2. プリフェッチ・スロットリング

2.1 既存のプリフェッチ量制御技術

既存の制御手法には大きく 2 通りのアプローチがあげられる．1 つはアドレス予測を行うごとに，その予測の正確さをさらに予測する手法である[10)]．トリガとなるロード命令の PC や参照アドレスをキーとして，過去の該当プリフェッチの成績を参照し，精度が低い場合にはリクエストをキャンセルする．アドレス予測における確信度に近いが，アドレス予測器と精度予測器のキーを柔軟に設定できる点が利点である．この方法では，過去のプリフェッチの履歴を保持するためのテーブルが新たに必要となる．

もう 1 つのアプローチは，一定期間ごとのプリフェッチ成績統計を動的に算出し，次の期間のプリフェッチ量へフィードバックする手法である[11)–13)]．プリフェッチ量の調整は，個々のプリフェッチリクエストを止めることではなく，プリフェッチャに設定された積極度を切り替えることによって行われ，このような制御はスロットリングと呼ばれることがある．一般に，プリフェッチの深さを増減させる積極度設定が用いられる．予測ごとに有効性を判断するアプローチに比べ，有効性判定の回数や必要な資源が少なく，近年の大量予測プリフェッチの制御に適している．

いずれのアプローチでも，プリフェッチがどれだけ無駄なラインを載せないかという，予測精度に着目することが多い．しかし，Ebrahimi らの研究[13)] では，プリフェッチによって回避されたキャッシュミスの数をカバー率（coverage）という統計で評価し，精度と合わせて用いることにより，さらに良い結果が得られることを示している．

2.2 既存スロットリング手法の課題

これらのスロットリングでは，プリフェッチの精度やカバー率を算出し，あらかじめ設定した閾値を上回れば積極度を加速，下回れば減速する．閾値を高くすれば抑制力に優れ，閾値を低くすれば加速に優れることとなる．しかし多様なワークロードに対して適切に働く閾値を一律に決めることは難しく，i) プリフェッチ精度は高いものの，キャッシュ容量に余裕がなく必要ラインを追い出してしまう場合，ii) 精度は低いもののキャッシュに余裕があり，プリフェッチによってミスを減らすことが有効な場合，iii) キャッシュ置き換えアルゴリズムが有効でなく，つねに新しいデータを載せた方が有効な場合などに，適切な制御ができないことが予想される．

3. Cache Convection

3.1 理想的なプリフェッチスロットリング

ここで，キャッシュミスを減らす適切なプリフェッチ・スロットリングについて考える．プリフェッチ・スロットリングは，キャッシュがデータを取り込むタイミングと量を変化させる．あるラインが通常よりも早くキャッシュに取り込まれるとき，入れ替わるラインは通常よりも早く追い出される．また，プリフェッチによって複数ラインが予測され読み込まれるとき，代わりに複数のラインが追い出される．読み込まれたラインの価値と追い出されたラインの価値の差がプリフェッチの利得である．ところが，従来手法の多くでは，プリフェッチによって載せられるラインの価値を詳細に評価する一方で，追い出されるラインの価値は判断せず，いわば一定に見積もっている．このことが，上に述べたような制御の齟齬を生み出してしまっている．

プリフェッチの目的は，ワーキングセットのメモリ内領域での移動が速く，キャッシュが

有効に機能しないアプリケーションにおいて，キャッシュ内の内容の入れ替えを早め，ワーキングセットの移動速度に追随させることであるといえる．速い移動に追随させるためには，遠くまで先読みしたり，複数のアドレスに網を張ったりすることが有効となる．一方で，入れ替えを早めすぎると，キャッシュの移動がプログラムを追い抜き，ポリューションが生じることとなる．つまり，最適なスロットリングは，まだラストユーズを終えていないラインを追い出さない範囲で，最大限積極的なプリフェッチを行うような手法である．

3.2　CC 値とスロットリング

キャッシュラインはキャッシュに取り込まれると，プログラムによってある回数参照された後，キャッシュのデータ取り込みに従って LRU 側へ移動し，追い出される．一般に，個々のラインのラストユーズを判断することは難しいが，一定期間のキャッシュ入れ替えを監視する場合，統計により特徴的な値を抽出することは可能である．以下では，この推定方法を述べ，スロットリング指標として使用可能であることを示す．

まず，まだラストユーズを終えていないキャッシュライン l について考えると，このラインが追い出されないためには，キャッシュエントリ数と等しい N 個のラインが追い出されるまでに，このラインへのアクセスが行われなければならない[⋆1]．アクセスがあれば，ラインは MRU 側へ復帰する．今 N 個のラインが追い出される間に h 回のキャッシュヒットがあったとすると，このラインへの統計的なアクセス回数は，

$$\frac{h}{N \times \mu} \tag{1}$$

と近似できる．ただし，μ はキャッシュ中の，ラストユーズを終えていないラインの割合である．キャッシュヒットは，キャッシュ全体が対象ではなく，これらのラインに対してのみ生じることに注目する．

式 (1) の値は 1 を超える場合もあり，そのフェーズで，ラインがどの程度繰り返しアクセスされるかの目安といえる．たとえば，プログラムがストリーム的にメモリの広い領域を走査する場合には低く，狭い領域にアクセスが集中する場合には高くなる．この値は，キャッシュラインが MRU 側と LRU 側を対流するイメージから，以降 **cache convection 値（CC 値）**と呼ぶ．

CC 値とプリフェッチ量の関係を考える．プリフェッチ量を増やしていくと，初期参照ミスが減少して，同じラインスワップ回数に対して h が増えるため値は増加する．CC 値は，

⋆1 統計なので，各セットは均等にアクセスされると仮定する．

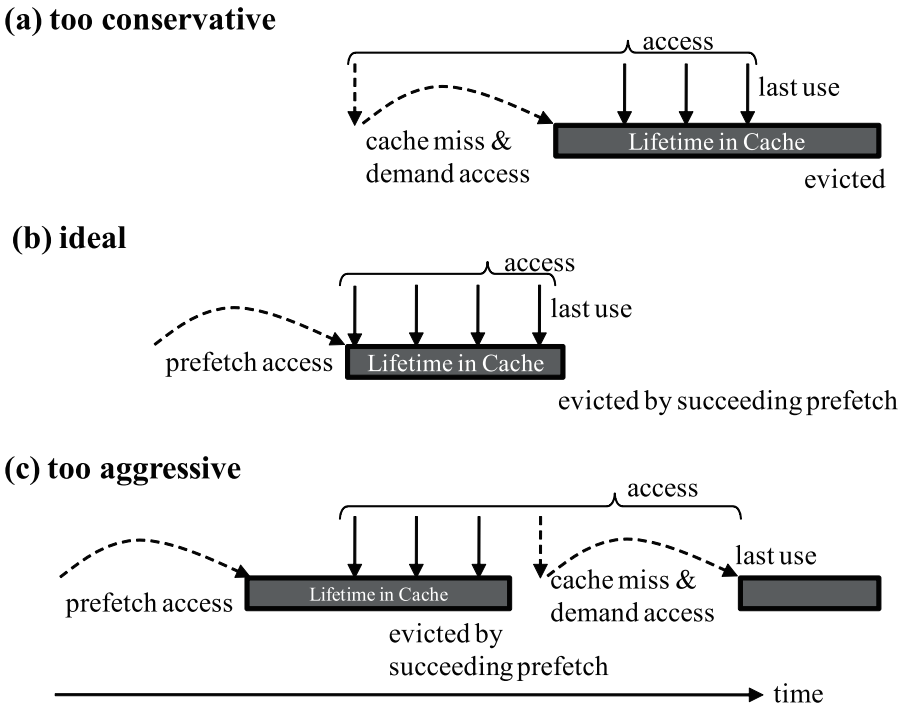

図 1　プリフェッチ積極度とライン再利用回数
Fig. 1　Prefetch aggressiveness and line reuse.

プログラムがどの程度ラインを再利用するかによって本質的な上限が決まっているため，プログラムフェーズごとに特有のある値に達すると増加は止まる．さらに積極度を上げていくと，必要となるタイミングよりも早い段階でキャッシュにラインが載ることになり，μ が増加して，CC 値は減少に転じる．さらに積極度が上がり，ポリューションが生じると，ラインスワップ回数が増加すると同時に h が減少するため，CC 値は大きく減少する．

この様子を，あるラインの寿命に注目して**図 1** (a)〜(c) に示す．横軸に時間をとったタイムラインとなっており，ラインがキャッシュに存在するライフタイムが黒いバーで示してある．(a) から (c) に移るに従って積極度が上がり，ラインの入れ替わり速度が速くなっている．CC 値のイメージはライフタイムのバー 1 つあたりのアクセスの本数である．(b) に示すように理想的なプリフェッチ流入量のとき，CC 値は同じフェーズの中で最も高い値と

なる．

この性質を用いて，CC 値を用いた最適スロットリング制御，CCCPO（Cache-Convection-Control based Prefetch Optimization）を提案する．一定のラインスワップ回数ごとに期間を区切り，その区間でのヒット数から CC 値を算出する．CC 値の増減と積極度の増減を対応させることにより，プリフェッチ流入量を最適に保つことができる．

4. CCCPO の実装

4.1 ハードウェア概要

CCCPO の概要を**図 2** に示す．一定回数のラインスワップが生じるごとに CC 値を算出し，その変化から，次の期間のプリフェッチ積極度を決定する．提案手法ではフィードバック用の履歴テーブルは不要であり，数本のレジスタ・カウンタで実装することができる．

追加されるカウンタは以下の 3 本であり，対象とするキャッシュについてアクセスやラインスワップが生じるごとに更新される．これらのカウンタは，各期間ごとにリセットされ，0 から計測を開始する．

- num_swap
- num_hit
- num_accessed

num_swap はキャッシュのラインスワップ数をカウントし，期間の区切りとする．num_hit はキャッシュのヒット回数をカウントする．num_accessed はアクセスのあったラインが追い出された数をカウントし，μ の推定に用いる（後述）．

追加されるレジスタは以下の 3 本であり，各期間の間，あるいは複数期間にまたがって値を保持する．

- reg_aggressivenes
- reg_previousCC
- reg_maxCC

reg_aggressivenes は現在のプリフェッチ積極度であり，プリフェッチャはこの値に従ってプリフェッチ量を変化させる．reg_previousCC には前期間の CC 値を保持する．reg_maxCC には近傍期間での最大 CC 値を保持する．

以上のレジスタ・カウンタのほか，提案手法では，CC 値の算出のために，キャッシュタグ上の 1 ビットのアクセスビットと，除算器を利用する．アクセスビットは，ラインがキャッシュに載るときに 0 で初期化され，その後そのラインにアクセスがあると 1 にセットされる．他の要素技術のためにすでに導入されている場合には共用できる．除算器に関して，本論文では専用の除算器を備えているものとして評価を行った．ただし，本手法で用いる除算器は，使用頻度が低く，精度は必要なく，また遅延も許容される性質のため，資源を節約する工夫が可能である．機能ユニット上の除算器を利用する，飽和カウンタを複数用いて近似する，などの方法が考えられる．

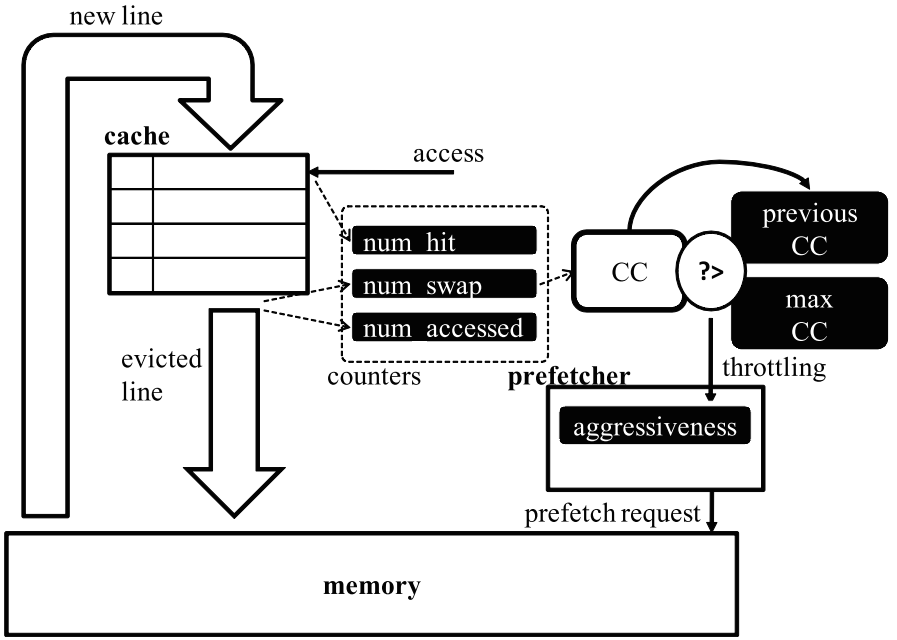

図 2　提案手法の概要
Fig. 2　Outline of the proposed technique.

4.2 CC 値の算出

追い出されたラインそれぞれについて，そのラインが 1 度でもアクセスされたかどうかを調べ，追い出しライン中の割合を計測する．アクセスの有無は，タグ中のアクセスビットを用いて判定する．1 度もアクセスされずに追い出されるラインは，以下のときに発生する．i) キャッシュミスによるデマンド・アクセスで読み込まれたが，初期アクセスがラストユーズであり，その後アクセスがなかった，ii) プリフェッチにより読み込まれたが，アクセスが

なかった．双方の場合とも，ラストユーズが終わった状態でキャッシュに載せられるケースである．このように，1 度もアクセスされないラインと，1 度以上アクセスされるラインとの比をとることによって，μ を近似する．

式 (1) の相対値は，カウンタを用いて，

$$\mathrm{rawCC} = \frac{\mathrm{num\_hit}}{\mathrm{num\_accessed}} \tag{2}$$

と算出することができる．

一定期間で区切って統計を用いるため，式 (2) の値は，本来の値の周りを細かく振動することが予想される．そこで，それまでの値と累積をとり，

$$\mathrm{CC} = \frac{\mathrm{rawCC} + \mathrm{reg\_previousCC}}{2} \tag{3}$$

をスロットリング指標として用いる．

4.3　制御アルゴリズム

3 章で示したように，同じフェーズ中では，CC 値を最大とする制御が最適である．num_swap が一定数に達するごとに以下の制御を行う．まず，式 (3) によって得られた期間 CC 値と，レジスタに保持してある前回の CC 値とを比較する．このときに CC 値が前回よりも一定割合以上減少していたら，プリフェッチ積極度を 1 段階減少させる．それ以外の場合には，プリフェッチ積極度を 1 段階増加させる．このように制御することによって最も高い CC 値を得る積極度を選択する．減少時にヒステリシスを適用するのは，累積同様，細かい振動の影響を排除するためである．積極度変更後，すべてのカウンタを 0 にリセットし，次の区間の計測を開始する．

この制御では，フェーズが変化した場合に CC 値の上限が変わることに注意が必要である．たとえば，ラインが何度も参照されるフェーズから，ライン参照回数が少なく，広いメモリ領域を走査するフェーズへ移った場合について考える．この場合，CC 値は高い状態から低い状態へ，フェーズ変化の前後で急激に移行する．フェーズ変化後は，プリフェッチ積極度を加速することが適当であるにもかかわらず，式 (3) の累積によって CC 値が高い値から徐々に減り続けるため，フェーズの先頭期間ではプリフェッチを減速する方向に制御が働いてしまう．このような変化は，プログラム中でしばしば発生する．

そこで，このようなフェーズ変化を検出するため，CC 値の最大値を reg_maxCC に保持する．算出された CC 値が，reg_maxCC の一定割合以下であった場合，フェーズ変化を検出し，CC 値の累積および reg_maxCC をリセットする．これにより，フェーズ変化後にすみやかにプリフェッチを加速させることができる．

5. 評 価 環 境

5.1　ベースラインプロセッサ

プロセッサシミュレータ鬼斬弐[14] にプリフェッチャ，提案スロットリング機構，メモリバスを実装し，サイクルレベルの評価を行った．プロセッサ命令セットおよびマイクロアーキテクチャパラメータは**表 1** に示した値を用いた．シングルスレッドシングルコア実行，プリフェッチは最下層キャッシュ（L2）へのデータアクセスにのみ適用した．L2 のみの適用とした理由は，プリフェッチの効果としてオフチップメモリレイテンシの隠蔽に注目しているためである．メモリレイテンシは，アクセスレイテンシ 200 サイクルと，メモリバスのバンド幅制限による待ち時間の合計としてモデル化している．メモリバスのバンド幅は，DPC-1[15] におけるレギュレーションと同等に，10 サイクルに 1 ラインの転送速度とした．なお，今回は DRAM バンク競合などは考慮しない，単純なモデルとなっている．

5.2　プリフェッチャ

スロットリング対象となるプリフェッチャとして，シーケンシャルプリフェッチャ[3] を用いた．キャッシュミスが発生すると，ミスアドレスから連続する領域にプリフェッチを行う．7 段階の積極度を**表 2** のように設定し，先読み量を変化させた．この設定は Ebrahimi ら[13] の用いた積極度に，さらに積極的な “level5”，“level6” を加えたものとなっている．

5.3　スロットリングパラメータ

CCCPO のパラメータとして，今回の評価では以下の値を用いた．まず，スロットリング調整は，L2 キャッシュラインのスワップが 2,048 回生じるごとに行う．減速の条件は，1

表 1　ベースラインプロセッサ緒元
Table 1　Microarchitectural parameters for baseline processor.

命令セット	Alpha ISA
フロントエンド	4way，7cycle
命令ウィンドウ	i64 entry，f32 entry
LSQ	32 entry
機能ユニット	2 iALU，1 iMUL/DIV，2 LD/ST，1 fpADD，1 fpMUL/DIV/SQRT
L1 ICache	32 KB，LRU，8way，64B line，1cycle latency
L1 DCache	32 KB，LRU，8way，64B line，1cycle latency
L2 I/DCache	256 KB，LRU，16way，64B line， 20 cycle latency
memory access	200 cycle + arbitration latency

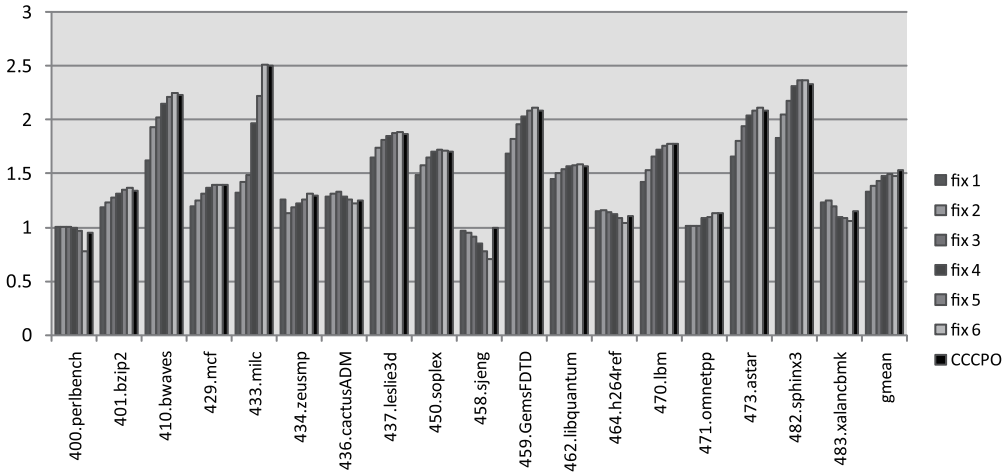

図 3　制御なしプリフェッチと CCCPO の性能比較（IPC）
Fig. 3　Performance comparison between fixed prefetch and CCCPO (IPC).

表 2　プリフェッチャ積極度設定
Table 2　Settings of prefetcher aggressiveness.

level0	none
level1	sequential depth 4
level2	sequential depth 8
level3	sequential depth 16
level4	sequential depth 32
level5	sequential depth 64
level6	sequential depth 128

つ前の CC 値からの 25%以上の低下とした．またフェーズ変化は，CC 値が近傍の最大 CC 値の 5%以下となったときに検出している．これらの値は，予備評価を行って決定したが，厳密なパラメータサーチの結果ではなく，最適化の余地は残っている．ただし，これらのパラメータの変化が性能に与えるセンシティビティは低い．

5.4　実行ベンチマーク

SPEC CPU2006 の全ベンチマークから，L2 ミスが性能に影響を与えているベンチマーク 18 本を選び，入力とした．具体的には，L2 容量およびプリフェッチ積極度を変化させて予備評価を行い，ベースラインに対し 20%以上の IPC 変化が生じたものを抽出した．各ベンチマークプログラムは gcc バージョン 4.2.2 を用いて O3 オプションでコンパイルしたものを用いた．シミュレーションでは，先頭 10 G 命令をスキップ後，その後の 100 M 命令について cycle accurate に実行して計測を行った．

6. 評　　価

6.1　提案スロットリング手法の効果

図 3 に，スロットリングを適用した場合と適用しなかった場合の性能を示す．“fix 1”～“fix 6” は，プリフェッチャの積極度を表 2 の “level 1”～“level 6” にそれぞれ固定した場合の性能を表している．また，“CCCPO” は提案手法によるスロットリングを適用した場合の性能を表している．それぞれ，プリフェッチを行わない場合からの相対 IPC で示している．また**図 4** にプリフェッチを行わなかったときを 1 とした相対ミス数を示す．

積極度と IPC の関係に注目すると，433.milc のように積極度とともに性能が増加するもの，逆に 458.sjeng のように積極度を増やすと性能がかえって低下するものなど，ベンチマークによって様々であることが分かる．積極度設定による性能変化は −30%から+250%までの幅がある．動的な制御にもかかわらず，提案手法は，この大きな性能変動の中で 483.xalancbmk を除き，つねに最高値に近い性能を示している．18 ベンチマークの調和平均で見た場合，

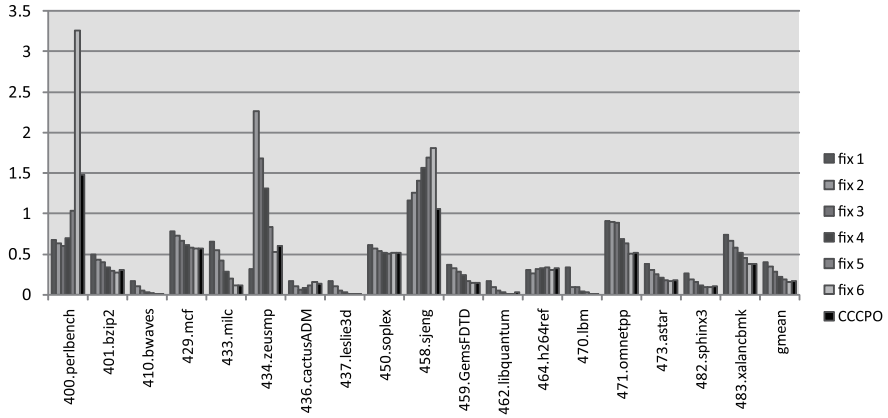

図 4　制御なしプリフェッチと CCCPO の性能比較（ミス数）
Fig. 4　Performance comparison between fixed prefetch and CCCPO (cache miss).

ベースラインに対し 53%，最も性能の高い固定積極度（“fix 5”）に対しても 3.7%の性能向上を得ている．

スロットリングの様子を**図 5** に示す．これは 458.sjeng について，実行中の様子を拡大したものである．横軸を共通とする，2 段のグラフとなっている．L2 が 2,048 回ラインスワップするごとに，下段にミス率（ただし，L1 ミス中の L2 ミス率），CC 値を，上段に積極度の変化をプロットしている．横軸はリタイア命令数であり，プログラムの実行ポイントに対応している．実行命令あたりのラインスワップ頻度が高いほど，密なグラフとなる．このグラフでは，積極度が 0 から 1 に上昇すると CC 値が下がり，それを受けて積極度が 0 に戻っており，積極度が最適値の周りで振動し，安定している様子が示されている．また，**図 6** に 450.soplex について同様の様子を示す．このグラフは 100 M の実行を通してすべてプロットした場合の例であり，プロット点が多く，各期間の制御の様子を追うことは困難だが，ミス率の周期的なパターンから，プログラムの特徴的なフェーズが繰り返されていることが分かる．積極度はフェーズ変化に合わせ，様々な値をとっている．

6.2　既存スロットリング手法との比較

既存のプリフェッチ・スロットリングでは，プリフェッチ精度を閾値と比較してフィードバックを行う．また，さらに高性能なフィードバック手法として，Ebrahimi ら[13] は，精度とカバー率の双方を用いるスロットリング手法を提案している．これらのスロットリング

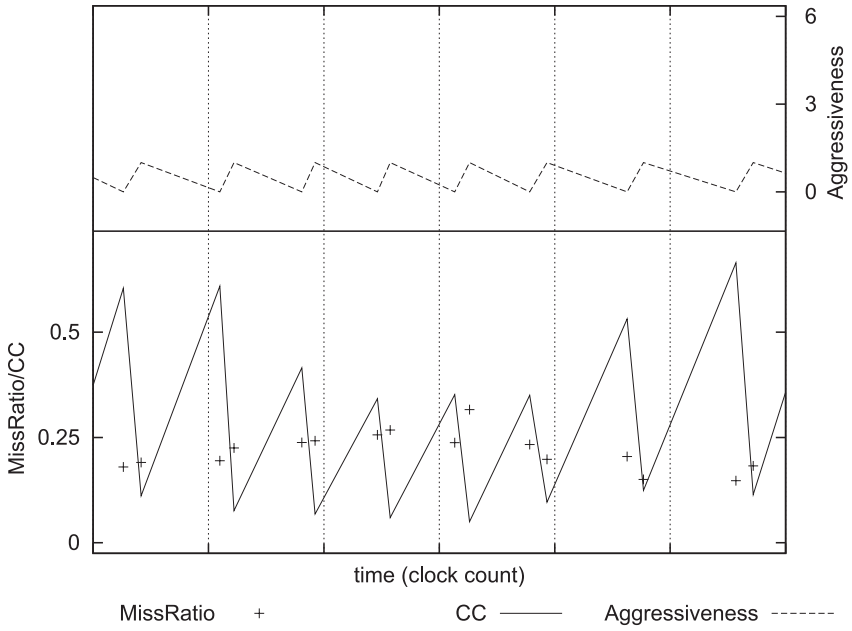

図 5　スロットリングの様子（458.sjeng）
Fig. 5　Throttling timeline (458.sjeng).

と，キャッシュ内データの対流に着目した CCCPO とを比較した．**図 7** に，各スロットリング手法による性能を示す．“Acc+Cov” が Ebrahimi らの手法[13] 同様，精度とカバー率の双方を用いる制御を示している．従来手法で問題となる，閾値のセンシティビティに関しては，パラメータサーチを行い最も性能の良い，“精度 80%以上またはカバー率 20%以上” を採用した．“Acc” は指標として精度のみを用いた場合で，“精度 80%以上”，となっている．条件を満たしたときに次の期間の積極度を 1 段階増加させ，満たしていなかった場合は 1 段階減少させる．“CCCPO” が提案手法である．また，参考として，“ideal” は各プログラムにおける最適積極度に固定した場合，すなわち，“fix 1”〜“fix 6” の中から最も良い値を選んだ場合の性能を表している．それぞれ，プリフェッチを適用しなかった場合からの相対 IPC で示している．

調和平均を見ると，提案手法は “Acc” に対して 10%，“Acc+Cov” に対して 1.5%高い性能を示している．さらに，個別のプログラムに注目すると，“Acc” および “Acc+Cov” では

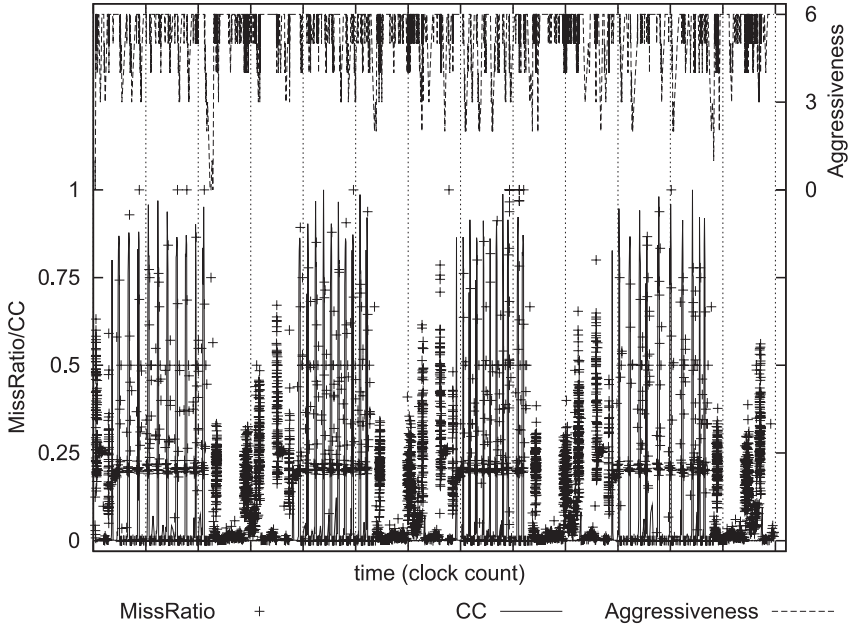

図 6　スロットリングの様子（450.soplex）
Fig. 6　Throttling timeline (450.soplex).

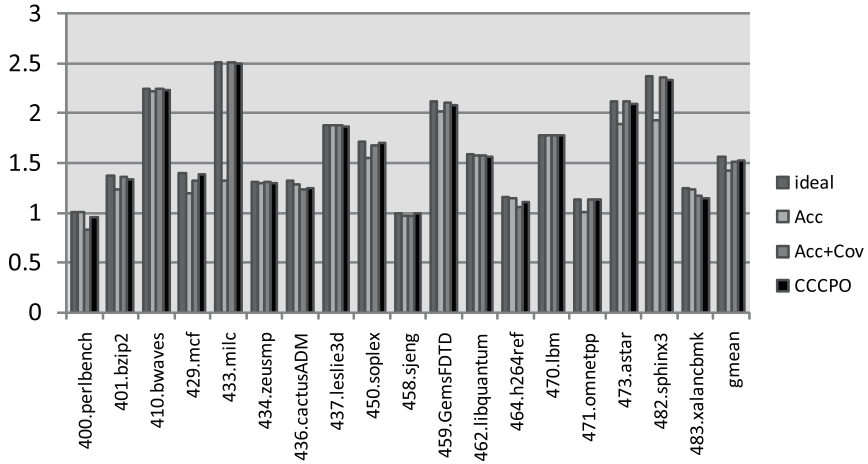

図 7　各スロットリング手法の性能比較（IPC）
Fig. 7　Performance comparison of prefetch throttling algorithms (IPC).

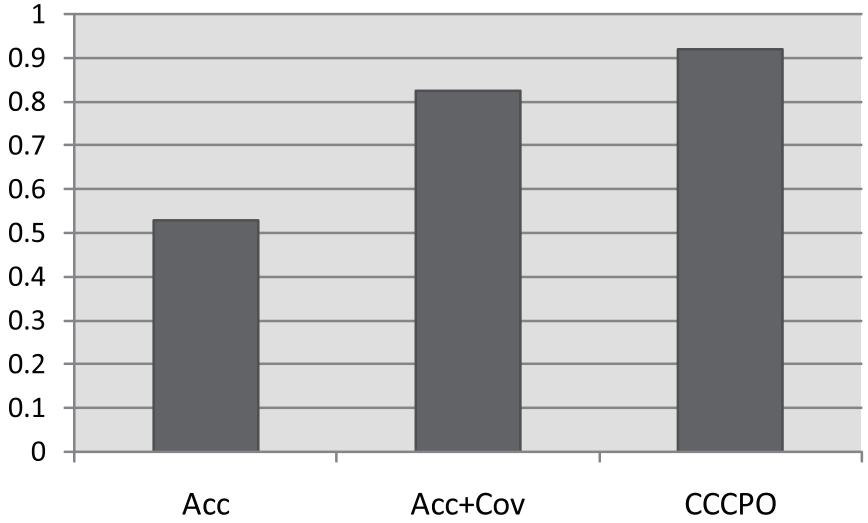

図 8　スロットリングの安定度（対 ideal）
Fig. 8　Reliability of the throttling (vs. ideal).

最適積極度から大きく離れた性能を示す場合があることが分かる．この様子を**図 8** に示す．このグラフは，ideal から最も乖離したときの IPC を ideal からの相対 IPC で示したものである．乖離は “Acc” では 50%近くにも及び，“Acc+Cov” においても，20%近い．CCCPO ではこの値は 10%程度であり，まだ改良の余地が残っているものの，従来に比べ安定した制御となっている．

これは，従来手法ではプリフェッチの有効性にのみ注目しているため，i) プリフェッチが当たるがキャッシュに余裕がないとき（400.perlbench など），ii) 精度は低いがプリフェッチを多くだせばミスを防げるとき（433.milc など），iii) プリフェッチの精度・カバレッジともに低いものの，キャッシュ置き換えアルゴリズムが役に立っていないとき（429.mcf），に正しい制御に失敗するためである．“Acc” はポリューションを起こすプログラムにおいて悪影響を防ぐブレーキ力に優れ，逆に “Acc+Cov” は積極的なプリフェッチが効果的なときの加速力に優れている．双方の判断が大きく分かれるプログラムが多く存在するが，このよう

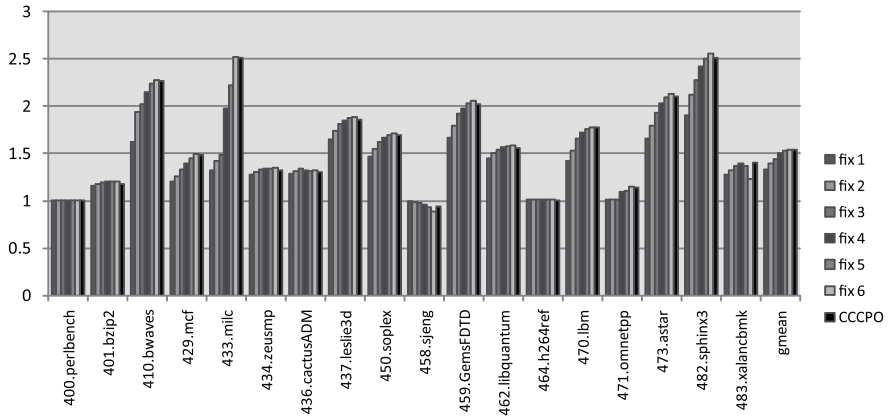

図 9　制御なしプリフェッチと CCCPO の性能比較（2 M）
Fig. 9　Performance comparison between fixed prefetch and CCCPO (2 M).

なプログラムにおいて提案手法は，483.xalancbmk を例外として，高い側に近い性能を示している．

このように，プリフェッチ・スロットリングでは，プリフェッチの有効性を追跡するよりも，キャッシュのデータ取り込み速度のバランスを調整する方が正確な制御となることが分かった．この手法は性能に優れるだけでなく，軽量なハードウェアで，また閾値の調整をすることなく，多様な環境へ適用することができる．異なるプリフェッチアルゴリズムやキャッシュ容量，バンド幅においても安定しており，たとえば**図 9** は L2 キャッシュ容量を 2 M としたときの相対 IPC を示している．キャッシュ容量が増加したことによりポリューションの影響が減り，全体的に高い積極度が有利となっているが，CCCPO の示す傾向は変わらず，最も高い調和平均を得ている．

7. 関 連 研 究

Hur ら[11] の adaptive stream では，期間中のストリーム長の統計を求め，最頻のストリームを算出し，その長さ以上にプリフェッチをかけない手法を提案している．このことにより，ストリーム終了時にプリフェッチがオーバランして生じる無駄を減らし，ストリーム長が短いプログラムでの効率を高めている．Srinath ら[12] は，各期間ごとにプリフェッチの精度，遅さ，ポリューションの 3 つの指標を算出し，その値をもとに次の期間の積極度を制御する手法を提案している．精度だけでなく，プリフェッチの速度やポリューションも折り込んだ制御となっているが，ポリューションの検知のために，追い出したラインの情報を格納する必要があり，キロビット単位のハードウェア資源を必要とする．Ebrahimi ら[13] は，Srinath らの手法を発展させ，カバレージと精度を併用するスロットリング手法を提案し，複数のプリフェッチャを併用するときのスロットリング手法として提案している．さらに Ebrahimi らは，プリフェッチ精度からフィードバックするスロットリング手法をマルチコアプロセッサに適用する手法[16] を提案している．プリフェッチ情報を用いないスロットリング手法として，Ramos ら[17] の手法では，一定サイクル中の L1 ヒット数を計測し，その増減を積極度の増減に反映させる．しかし，指標がサイクルあたりのヒット数の増減であるため，フェーズによって制御が乱されることが考えられる．また，プリフェッチは過剰でも過少でもヒット数を減らしてしまうため，ヒット数の増減をそのまま積極度の増減とするこの手法では制御が安定しない場合があると考えられる．

プリフェッチの積極度を上げる方法として，深さを増やすほかに，複数の予測アルゴリズムを組み合わせるアプローチがある．我々[18] はフェーズごとに有効なアルゴリズムをイネーブルするプリフェッチ制御法，SHPF を提案している．また，Ishii ら[9] の AMPM プリフェッチは複数のストライド候補を同時に発見し多くのアドレスにプリフェッチをかける．この手法では予測アルゴリズムにポリューションやアクセス速度などを見込んだ制御が組み込まれ，性能を最適化している．

8. お わ り に

ハードウェアプリフェッチは，メモリレイテンシを効率良く隠蔽する技術だが，効果的に用いるためには，適切な積極度制御が欠かせない．本研究では，プリフェッチの成績ではなく，キャッシュ状態に着目するプリフェッチ・スロットリング手法を提案した．ライン再利用回数は，プログラムの振舞いによってフェーズごとに特徴的な値を持つことを利用し，これを推定するための Cache-Convection 値を指標として提案した．CC 値は，プログラム進行によるワーキングセットの推移と，キャッシュのデータ取り込み速度が釣り合ったときに最大となり，この状態は，最適なプリフェッチ積極度にほかならない．

CC 値を利用してプリフェッチスロットリングを行う CCCPO を提案，評価した．CCCPO は，多様なワークロードバランスに対して安定したスロットリングを行うことができる．従来手法のようにセンシティブな閾値を探索する必要がない．さらにプリフェッチ有効性の追跡を行わず，数本のカウンタ・レジスタで実装可能である．シミュレータによる評価では，

最新のスロットリング手法に対して，最大で 13%，調和平均でも 1.3%の性能向上を示した．今後の課題として，マルチコアにおける共有キャッシュへのプリフェッチスロットリングや，ハイブリッドプリフェッチャのスロットリングがあげられる．

謝辞　本研究のデータは東京大学情報理工学系研究科 istbs クラスタ上で取得されました．クラスタ環境にご尽力いただいた，東京大学の田浦健次朗先生，同研究室の原健太朗様に深く感謝の意を表します．

参 考 文 献

1) Patterson, D.: Latency Lags Bandwidth, *Comm. ACM*, Vol.47, No.10, pp.71–75 (2004).
2) Smith, A.: Sequential Program Prefetching in Memory Hierarchies, *IEEE Computer*, Vol.11, No.12, pp.7–21 (1978).
3) Jouppi, N.: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, *Int. Symp. on Computer Architecture*, pp.364–373 (1990).
4) Gebis, J. and Patterson, D.: Embracing and Extending 20th-Century Instruction Set Architecture, *IEEE Computer*, Vol.40, No.4, pp.68–75 (2007).
5) Tendler, J., Dodson, J., Fields, J., Lee, H. and Sinharoy, B.: Power4 system microarhitecture, *IBM Journal of Research and Development*, Vol.46, No.1, pp.5–26 (2002).
6) Cooksey, R., Jourdan, S. and Grunwald, D.: A stateless, content-directed data prefetching mechanism, *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.279–290 (2002).
7) Nesbit, K. and Smith, J.: Data Cache Prefetching Using a Global History Buffer, *Int. Symp. on High-Performance Computer Architecture*, pp.96–105 (2004).
8) Somogyi, S., Wenisch, T., Ailamaki, A. and Falsafi, B.: Spatio-Temporal Memory Streaming, *Int. Symp. on Computer Architecture*, pp.69–80 (2009).
9) Ishii, Y., Inaba, M. and Hiraki, K.: Access map pattern matching for data cache prefetch, *Int. Conf. on Supercomputing*, pp.499–500 (2009).
10) Zhuang, X. and Lee, H.: A Hardware-based Cache Pollustion Filtering Mechanism for Aggressive Prefetches, *Int. Conf. on Parallel Processing*, pp.286–293 (2003).
11) Hur, I. and Lin, C.: Memory Prefetching Using Adaptive Stream Detection, *Int. Symp. on Microarchitecture*, pp.397–408 (2006).
12) Srinath, S., Mutlu, O., Kim, H. and Patt, Y.: Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers, *Int. Symp. on High-Performance Computer Architecture*, pp.63–74 (2007).
13) Ebrahimi, E., Mutlu, O. and Patt, Y.: Techniques for Bandwidth-Efficient Prefetching of Linked Data Structures in Hybrid Prefetching Systems, *Int. Symp. on High-Performance Computer Architecture*, pp.7–17 (2009).
14) 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム 2009 ポスター (2009).
15) JILP Data Prefetching Championship. http://www.jilp.org/dpc/
16) Ebrahimi, E., Mutlu, O., Lee, C. and Patt, Y.: Coordinated Control of Multiple Prefetchers in Multi-Core Systems, *Int. Symp. on Microarchitecture*, pp.316–326 (2009).
17) Ramos, L., Briz, J., Ibanez, P. and Vinals, V.: Multi-level Adaptive Prefetching based on Performance Gradient Tracking, *Workshop on JILP Data Prefetching Championship* (2009).
18) 本城剛毅，石井康雄，入江英嗣，稲葉真理，平木　敬：フィードバックを用いたハイブリッドプリフェッチング方式，情報処理学会研究報告 2009-ARC-184，No.18 (2009).

(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 5 日採録)

入江　英嗣（正会員）

1999 年東京大学工学部電子情報工学科卒業．2004 年同大学院情報理工学系研究科電子情報学専攻博士課程修了．博士（情報理工学）．2004 年科学技術振興機構 CREST 研究員．2008 年東京大学情報理工学系研究科助教．2010 年電気通信大学情報システム学研究科準教授．コンピュータ・システムの研究に従事．電子情報通信学会コンピュータシステム研究会幹事補佐（2007 年〜）．電子情報通信学会，IEEE，ACM 各会員．

本城　剛毅

2009 年東京大学理学部情報科学科卒業．同年同大学院情報理工学系研究科修理課程進学．プロセッサアーキテクチャの研究に従事．IEEE，ACM 各会員．

平木　　敬（正会員）

東京大学理学部物理学科，同大学院理学系研究科物理学専門課程博士課程退学，理学博士．工業技術院電子技術総合研究所，米国 IBM 社 T.J.Watson 研究センターを経て現在東京大学大学院情報理工学系研究科勤務．数式処理計算機 FLATS，データフロースーパコンピュータ SIGMA-1，大規模共有メモリ計算機 JUMP-1 等多くのコンピュータシステムの研究開発に従事，現在は超高速ネットワークを用いる遠隔データ共有システム Data Reservoir システムの研究，超高速計算システム GRAPE-DR の研究を行っている．